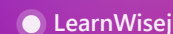


Arquitectura y vocabulario de integración

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Comprende el modelo componente/widget/control y el pipeline de renderizado y eventos de Wisej.NET. Qué significa esto para un desarrollador de integraciones con Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Comprende el modelo componente/widget/control y el pipeline de renderizado y eventos de Wisej.NET.

1. El problema de integración

Los desarrolladores suelen tratar los widgets de JavaScript como objetos aislados del navegador. En Wisej.NET, la integración debe diseñarse como un componente propiedad del servidor que renderiza estado, recibe eventos y delega el trabajo específico del navegador en un widget de cliente.

2. Conceptos clave

Concepto	Cómo usarlo
Component	Objeto .NET de servidor que posee estado y participa en el renderizado.
Widget	Objeto JavaScript del navegador que crea y actualiza el DOM o la instancia del widget.

3. Lección de producción

La integración con Wisej.NET parte de una regla de arquitectura sencilla: el servidor es dueño del estado de la aplicación, mientras que el navegador es dueño del DOM y de la instancia del widget de terceros. Un componente Wisej.NET envía una representación compacta de su estado relevante a un widget de cliente. Ese widget de cliente crea, configura, actualiza y conecta el objeto JavaScript de terceros. Cuando ocurre algo importante en el cliente, el widget de Wisej.NET envía de vuelta al componente de servidor una carga útil de evento. Este modelo es la misma idea en la que se basan los controles integrados de Wisej.NET. La diferencia es que, en las integraciones de terceros, el widget de cliente suele convertirse en un adaptador alrededor de otra biblioteca de JavaScript. El adaptador recibe el estado desde .NET, lo traduce al objeto de opciones o a las llamadas a métodos del proveedor, escucha los eventos del proveedor y reenvía al servidor solo la información significativa. Una buena integración no intenta ocultar que el control de terceros vive en el navegador. Mantiene el objeto del proveedor dentro del adaptador de cliente, expone solo el estado y los eventos que el lado .NET necesita de verdad y documenta ese contrato para que el servidor nunca tenga que adivinar qué está haciendo el widget.

El vocabulario, palabra por palabra

Las conversaciones sobre integración se tuercen cuando dos desarrolladores usan la misma palabra para cosas distintas. Un Component es cualquier objeto .NET de servidor que Wisej.NET sabe renderizar en el navegador; un temporizador o un origen de datos son Components sin superficie visual. Un Control es un Component que ocupa espacio en una página y tiene una representación visual. Un Widget es el objeto JavaScript del navegador que dibuja esa superficie y reacciona al usuario. Todo Control es un Component, pero no todo Component es un Control, y ninguno de los dos es el Widget.

Server y Client nombran los dos extremos de la conexión. Rendering es el momento en que el servidor serializa el estado relevante de un Component y lo envía a su Widget. Un Event es el momento en que el Widget envía algo de vuelta. State es todo lo que el servidor conserva entre esos dos momentos: valores de propiedades, selección, datos y permisos. En el `TemperatureGauge` del laboratorio, `Value` es estado del servidor, el indicador del proveedor es el Widget y `ThresholdExceeded` es un Event.

Usa estas palabras con precisión en los registros de decisiones y en las revisiones de código. Si una frase no puede escribirse con ellas, probablemente el diseño no está claro.

JSON de ida, JSON de vuelta

El pipeline de renderizado es sencillo a propósito. Cuando cambia el estado de un Component, Wisej.NET recopila las propiedades modificadas, las serializa como un objeto JSON compacto y entrega ese objeto al Widget. El Widget aplica cada valor: en un control integrado eso significa actualizar su propio DOM, y en una integración significa traducir el valor al objeto de opciones del proveedor o a una llamada a un método del proveedor. El servidor nunca toca el DOM y el navegador nunca guarda el estado de referencia.

El camino de vuelta tiene la misma forma, pero a la inversa. Cuando el Widget decide que algo importa, envía al servidor un evento con nombre y una pequeña carga útil JSON, y el servidor lanza un evento .NET en el Component. En el laboratorio, asignar 104 a `Value` en el servidor renderiza un pequeño cambio JSON hacia el indicador y, cuando el indicador supera su umbral, vuelve un único evento con el valor actual y `ThresholdExceeded` se dispara en C#.

Todo lo que cruza la conexión son datos, nunca comportamiento. Eso es lo que mantiene la integración revisable: el JSON en ambas direcciones puede registrarse, inspeccionarse en las herramientas del navegador y compararse con un contrato escrito.

Por qué el objeto del proveedor vive dentro de un contenedor

Un Widget de Wisej.NET posee un elemento DOM que el framework posiciona, dimensiona, muestra, oculta, tematiza y elimina. Una biblioteca de terceros también quiere poseer un elemento: crea hijos dentro de él, le asocia manejadores y a veces lo sustituye. Si al objeto del proveedor se le entregara el propio elemento de Wisej.NET, los dos propietarios chocarían. Los cambios de diseño se perderían, la eliminación dejaría manejadores huérfanos y una llamada del proveedor que reconstruyera el elemento rompería la contabilidad interna del framework.

Alojar el objeto del proveedor dentro de un elemento contenedor que proporciona el Widget evita el conflicto. El framework conserva su elemento, el proveedor conserva su propio subárbol por debajo y el Widget media entre ambos. El redimensionado se convierte en una llamada al método de redimensionado

del proveedor, la eliminación en una llamada a su método destroy seguida de vaciar el contenedor, y la tematización en un estilo aplicado al contenedor.

La excepción es una biblioteca diseñada para mejorar un único elemento en su sitio, como un campo de entrada con máscara. Incluso entonces, la mejora se aplica a un elemento dentro del contenedor, nunca a la raíz del framework.

Elegir la profundidad de la integración

No todos los widgets merecen la misma inversión. Una integración puntual coloca un `Wisej.Web.Widget` en una página con un script y unas opciones; es la adecuada para un prototipo, un dashboard interno o un widget que se usa una sola vez. Una integración reutilizable envuelve ese mismo widget en una clase con propiedades tipadas para poder colocarlo en muchas pantallas sin copiar la configuración. Una integración con soporte de temas va más allá y conecta la apariencia del widget con el tema de la aplicación. Una integración con aspecto nativo es un control personalizado completo, con clase de servidor y clase de cliente, que se diseña, se tematiza y se maneja por eventos como un control integrado.

El laboratorio toma esta decisión para tres candidatos. Un pequeño indicador usado en un solo dashboard es una integración puntual. Un editor de texto enriquecido que aparece en varios formularios y debe coincidir con el tema merece un wrapper reutilizable y con soporte de temas. Una cuadrícula de datos que se convierte en la superficie estándar de listas de la aplicación justifica un control con aspecto nativo, porque de ella dependen decenas de pantallas y años de mantenimiento.

- Cuántas pantallas lo usarán, ahora y dentro de dos años.
- Si debe coincidir con el tema y funcionar en el Designer.
- Quién mantiene la versión de la biblioteca del proveedor y sus cambios incompatibles.
- Cuánto cuesta equivocarse de profundidad más adelante.



Regla de producción

Mantén claras la propiedad del estado en Wisej.NET y el comportamiento del widget de cliente. Documenta el contrato cliente/servidor antes de confiar en la integración en producción.



Error habitual que debes evitar

Copiar JavaScript de demostración en las pantallas sin documentar el ciclo de vida, los recursos, los contratos de eventos, la limpieza ni los límites de seguridad.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Distinguir Component, Control, Widget, Client, Server, Rendering, Event y State.
- ✓ Explicar cómo los componentes de servidor renderizan estado JSON hacia los widgets de cliente y reciben de vuelta datos de eventos en JSON.

- ✓ Describir por qué los widgets de terceros suelen vivir dentro de un contenedor de widget de Wisej.NET en lugar de sustituir al propio widget de Wisej.NET.
- ✓ Decidir si un widget debe ser puntual, reutilizable, con soporte de temas o con aspecto nativo.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el flujo de trabajo recomendado, los entregables requeridos y la autoevaluación antes de entregar.



Laboratorio práctico

Crea un registro de decisiones de integración para tres widgets candidatos: un pequeño indicador (gauge), un editor enriquecido y una cuadrícula de datos. Decide qué tipo de integración merece cada uno y explica por qué.

Flujo de trabajo recomendado

- ✓ Abre el proyecto existente del curso de integración de Wisej.NET y ejecútalo en local.
- ✓ Identifica el componente de servidor, el widget de cliente, el archivo de recursos o el endpoint que intervienen en este módulo.
- ✓ Haz un solo cambio pequeño cada vez y pruébalo en el navegador antes de continuar.
- ✓ Mantén el JavaScript específico del proveedor dentro del adaptador o wrapper en lugar de repartirlo por las páginas.
- ✓ Actualiza la interfaz para que el estudiante vea con claridad el éxito, los errores, el progreso o los cambios de comunicación.
- ✓ Toma capturas de pantalla o notas que demuestren que la integración funciona y expliquen la decisión de diseño.
- ✓ Si usas un asistente de programación con IA, explícale el contrato y las reglas de Wisej.NET que verás más abajo antes de pedirle código; el Ejercicio de programación con IA de este módulo te guía por todo el proceso de principio a fin.

Entregables requeridos

- ✓ Diagrama de arquitectura de la integración
- ✓ Tabla de clasificación de widgets
- ✓ Primer ADR: enfoque de integración elegido para cada widget candidato

Autoevaluación antes de entregar

- ✓ ¿Por qué un Control es siempre un Component, pero no todo Component es un Control?
- ✓ ¿Por qué la mayoría de los widgets de terceros deberían alojarse dentro de un elemento contenedor?
- ✓ ¿Qué estado pertenece al servidor y qué estado puede quedarse en local en el navegador?

Trabajar con un asistente de programación con IA (Claude, OpenAI)

Claude, ChatGPT y los asistentes de programación basados en ellos (Claude Code, Codex, Copilot) escribirán esta integración rápidamente, pero una petición sin contexto te dará una respuesta genérica: un `System.Windows.Forms.Timer`, un widget creado directamente sobre `this.container`, un evento

lanzado con `fireWidgetEvent` desde dentro de `update()` o una sentencia `return` dentro de `EvalAsync`. Todo eso compila o se analiza sin errores, y todo falla en tiempo de ejecución. Dale al asistente el contrato y las reglas del framework desde el principio y esos problemas desaparecen. El asistente es un par de manos rápidas; tú sigues siendo el arquitecto que es dueño del contrato y verifica cada afirmación.

Explícale el trabajo como lo harías con un nuevo miembro del equipo

- ✓ **Primero, el vocabulario.** Deja claro que un Component es un objeto .NET de servidor que posee estado, que un Control es un Component con superficie visible y que un Widget es el adaptador del navegador. Decir «Widget» sin esta definición invita a escribir código solo de cliente.
- ✓ **Las reglas del framework en el primer mensaje.** Pégalas literalmente: la clase de servidor deriva de `Wisej.Web.Widget`; los scripts del proveedor son `Packages`; el adaptador es el `InitScript` cargado con `GetString` desde un recurso incrustado; el estado se escribe en `((dynamic)this.Options)` y llega como `init(options) / update(options, old)`; los eventos de cliente se declaran en `WiredEvents`, se conectan en el cliente mediante `_addListener / _getEventData` y llegan a `OnWidgetEvent`.
- ✓ **El contrato antes que el código.** Dale al asistente los campos de estado, los nombres y las cargas útiles de los eventos y las rutas de error. Pídele que reformule el contrato con sus propias palabras antes de escribir una sola línea; corregir una reformulación equivocada sale más barato que corregir código equivocado.
- ✓ **Indica las versiones.** «Wisej.NET 4, NuGet Wisej-4 4.1.0, net10.0, host Kestrel» elimina toda una categoría de respuestas pensadas para Wisej 2 / .NET Framework.

Pide piezas pequeñas y revisa cada una

- ✓ Pide primero solo la clase de servidor, revísala y después pide solo el adaptador de cliente. Un archivo por turno mantiene la revisión honesta y los errores localizados.
- ✓ Di qué *no* debe cambiar: «mantén el contrato; no añadas eventos; no valides en JavaScript». Los asistentes amplían un diseño sin reparos mientras corrigen un error.
- ✓ Pega la salida del compilador y los errores de la consola del navegador **literalmente**. Los errores parafraseados llevan a correcciones adivinadas.
- ✓ Cuando una API no te resulte familiar, pregunta: «¿Qué clase y qué miembro de Wisej.NET es este? ¿Dónde está documentado?». Contrasta la respuesta con la documentación de Wisej.NET o con la documentación XML del framework antes de aceptarla.
- ✓ Pide las rutas de error de forma explícita: «¿Qué ocurre cuando el proveedor lanza una excepción dentro de `update()`? Muestra la ruta de código que la convierte en un único evento `error`». Si no lo pides, cualquier primer borrador se queda en el camino feliz.

Verifica y conserva lo que aprendiste

- ✓ Nada está terminado hasta que compila y las cinco rutas se ejecutan en el navegador: éxito, progreso, un valor no válido, una carga útil dañada y la recuperación. El panel de trazas es tu evidencia.

- ✓ Guarda una nota breve con las convenciones y correcciones que hayas acordado, y pégala en el primer mensaje de tu próxima sesión. Esa nota vale más que cualquier prompt aislado.
- ✓ Mantén el JavaScript específico del proveedor dentro del adaptador aunque el asistente se ofrezca a incrustarlo en la página; la misma regla vale para el código escrito por personas y por la IA.
- ✓ Nunca pegues claves de licencia, cadenas de conexión ni datos de clientes en un prompt. Usa marcadores de posición y una biblioteca de proveedor sustituta.



Reglas de Wisej.NET que debes indicar en las instrucciones

Los eventos lanzados de forma síncrona desde dentro de `update()` se descartan: conéctalos mediante `_addListener`. Definir `this.getWidth`, `this.getHeight` o `this.destroy` en el wrapper oculta un método de qooxdoo y puede reducir el widget a 0 px. `EvalAsync` recibe una expresión, no una sentencia `return`. Los objetos anónimos asignados a `Options` se transmiten en camelCase; los valores de retorno de un WebMethod, no. Usa `Wisej.Web.Timer`, nunca `System.Windows.Forms.Timer`.

La sección **Ejercicio de programación con IA** de este módulo pone todo esto en práctica: te entrega un prompt ya preparado que dirige un asistente a esta guía, y nuestro propio Sensor Monitor —creado exactamente de esta forma— necesitó dos correcciones tras ejecutarlo, ambas descritas arriba.